VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data

Hank Childs[*,§], Eric Brugger[†], Brad Whitlock[†], Jeremy Meredith[‡], Sean Ahern[‡], Kathleen Bonnell[†], Mark Miller[†], Gunther Weber[*], Cyrus Harrison[†], David Pugmire[‡], Thomas Fogal[¶], Christoph Garth[§], Allen Sanderson[¶], E. Wes Bethel[*], Marc Durant[‖], David Camp[*,§], and Jean M. Favre[▽]

[*]Lawrence Berkeley National Laboratory, [†]Lawrence Livermore National Laboratory, [‡]Oak Ridge National Laboratory
[§]University of California at Davis, [¶]University of Utah, [‖]Tech-X Corporation, [▽]Swiss National Supercomputing Centering

ABSTRACT

VisIt is a popular open source project for visualizing and analyzing data. It owes it success to its foci of data understanding, large data support, and providing a robust and usable product, as well as it underlying design that fits today's supercomputing landscape. In this short paper, we describe the VisIt project and its accomplishments.

1 INTRODUCTION

This is V8, Tuesday at 12:05

A dozen years ago, when the VisIt project started, a new high performance computing environment was emerging. Ever increasing numbers of end users were running simulations and generating large data. The rapidly growing number of large data sets prevented visualization experts from being intimately involved in the visualization process; it was thus necessary to put tools in the end users hands. Almost all end users were sitting in front of high-end desktop machines with powerful graphics cards. But their simulations were being run on remote, parallel machines and generating data sets too large to be transferred back to these desktops. Worse, these data sets were too large to even process on a single serial machine. Further, the types of visualization and analysis users wanted to perform varied greatly; users needed many techniques for understanding diverse types of data, with use cases ranging from confirming that a simulation was running smoothly to communicating the results of a simulation to a larger audience to gaining insight via data exploration.

VisIt was developed in response to these emerging needs. It was (and is) an open source project for visualizing and analyzing extremely large data sets. The project has evolved around three focal points: (1) enabling data understanding, (2) scalable support for extremely large data, and (3) providing a robust and usable product for end users. In turn, these focal points have made VisIt very popular for visualizing and analyzing the data sets from the world's largest supercomputers. VisIt has been downloaded hundreds of thousands of times and received a 2005 R&D100 award for the tool's capabilities in understanding large data sets.

This short paper aims to explore the VisIt project's focal points (§2), design (§3), and successes (§4).

2 FOCAL POINTS

Enable data understanding: A common misnomer about VisIt is that it is a tool strictly for making pretty pictures. Its unfortunate choice of name, now solidified through name recognition, pigeonholes the listener into thinking it is dedicated solely to visualization. In reality, VisIt focuses on five primary use cases:

1. *Visual exploration*: the user applies a series of visualization algorithms to "see" what is in their data.
2. *Debugging*: the user applies algorithms to find a "needle in a haystack," for example hot spots in a scalar field or cells that have become twisted over time. The user then asks for debugging information in a representation that is recognizable to their simulation (e.g. cell **X** in computation domain **D** has a NaN).
3. *Quantitative analysis*: the quantitative capabilities range from simple operations, such as integrating densities over a region to find its mass, to highly sophisticated operations, such as adding a synthetic diagnostic to compare to experimental data.
4. *Comparative analysis*: the user compares two related simulations, two time slices from a single simulation, simulation and experiment, etc. The taxonomy of comparative analysis has three major branches, each of which is available in VisIt: Image-level comparisons place things side-by-side and has the user detect differences visually. Data-level comparisons put multiple fields onto the same mesh, for example to create a new field that contains the difference in temperature between two simulations for further analysis. Topological-level comparisons detect features in the data sets and then allow those features to be compared.
5. *Communication*: the user wants to communicate properties of their data to a large audience. This may be via movies, via images that are inserted into a PowerPoint presentation, or via one dimensional curves that are placed into a journal article.

Support for large data: The definition of "large" is relative to the resources for processing the data. For the VisIt project, the target was data whose full resolution could not fit into primary memory of a desktop machine. Of course, the amount of data to load varies by situation: can we process time slices one at a time? how many variables do we need to load? do we need to load multiple members of an ensemble simultaneously? For VisIt, the goal was to provide an infrastructure that could support any of these use cases, and it primarily uses parallelism to achieve this goal.

Provide a robust and usable product for end users: Enabling data understanding for large data is a daunting task requiring a substantial investment. To amortize this cost, the project needed to be delivered to many user communities, across both funding groups and application areas.

The "one big tool" strategy provides benefits to both users and developers. Compared to a smaller, tailored effort, users are able to access more functionality as well as better underlying algorithms for processing data. For developers, the core infrastructure undergoes an economy of scale, where many developers can collectively develop a superior core infrastructure than they would be able to independently. But the "one big tool" approach has negative aspects as well. Users are provided an overly rich interface where many features may be meaningless to them and simply serve as clutter. And developers must deal with a less nimble code base where making functionality changes sometimes leads to unexpectedly large coding efforts.

Further, delivering a product to a large end user community incurs significant cost in and of itself: the VisIt project has almost a thousand pages of manuals, several thousand regression tests that

are run every night, a sophisticated build process, and a variety of courses designed to teach people to how to use the tool. It requires multi-institution coordination for release management, for responses to user requests, and for software development. And, of course, the source code itself must be well documented to ease barriers to entry for new developers.

The developers of the VisIt project decided to "go big": to pay the costs associated with large user and developer bases in the hopes of writing a tool that would be usable by many and developed by many.

3 DESIGN

In this section, we describe several facets of VisIt's design: §3.1 describes VisIt's architecture, §3.2 describes its parallelism approach, and §3.3 describes its user interface concepts.

3.1 Architecture

VisIt employs a client/server design, where both client and server are made up of multiple programs (see Figure 1). Client-side programs, typically run on the user's local desktop, are responsible for both user interface and rendering, where interactivity is paramount. The client-side programs are:

- **gui:** A graphical user interface built using the Qt widget set.
- **cli:** A command line user interface built using the Python language.
- **viewer:** A program responsible for visual display of data.
- Custom, streamlined user interfaces can also be added to VisIt. The interfaces can either complement the gui and cli or replace them altogether.

Server-side programs, typically run on a remote supercomputer that can access the user's data in a parallel fashion, are responsible for processing data. The server-side programs are:

- **engine:** The program that applies visualization and analysis algorithms to large data using parallel processing.
- **mdserver:** A program that browses remote file systems and reads meta-data.
- **vcl:** **V**isIt **C**omponent **L**auncher, a program whose sole job is to launch other programs. Without this program, the user would have to issue credentials for the launch of each program on the remote machine.

While the configuration in Figure 1 is the most common, other variants are also used:

- Data is located on the local machine, so all programs, including the server-side programs, run on the local machine.
- The client-side programs run on a remote machine. This mode occurs most often in conjunction with graphical desktop sharing, such as VNC.
- Multiple servers are run simultaneously to access data on multiple remote machines.
- VisIt is run entirely in "batch mode." The **gui** program is not used and the **viewer** program runs in a windowless mode.
- VisIt's client-side programs are coupled with a simulation code and data is processed *in situ*. In this case, the simulation embeds a copy of the **engine** program.

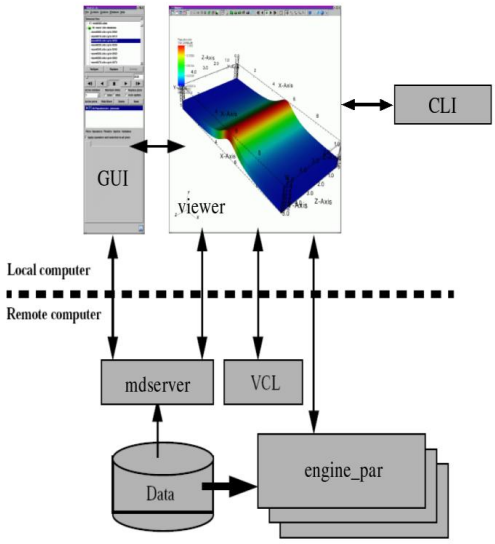

Figure 1: Diagram of VisIt programs and their communication.

3.2 Parallelism

VisIt supports multiple processing modes – multi-resolution processing, in situ processing, and out-of-core processing – but its most frequent mode is pure parallelism, where the data is partitioned over its MPI tasks and is processed at its native resolution. Most visualization and analysis algorithms are embarrassingly parallel, meaning that portions of the data set can be processed in any order and without coordination and communication. For this case, VisIt's core infrastructure manages the partitioning of the data and all parallelism. For the non-embarrassingly parallel case, for example streamline calculation or volume rendering, algorithms are able to manage the parallelism themselves and can opt to perform collective communication if necessary.

VisIt's most typical visualization use case has a user loading a large data set, applying operations to reduce the data size, and then transferring the resulting data set to the local client for interactive rendering using the local graphics cards. However, some data sets are so large that their reduced forms are too large for a desktop machine. This case requires a backup plan. VisIt's backup plan is to switch to a parallel rendering mode: data is left on the parallel server, each MPI task renders its own piece, and the resulting sub-images are composited together. The final image is then brought back to the **viewer** and placed in the visualization window, as if it was rendered with the graphics card. Although this process sounds cumbersome, switching to parallel rendering mode is transparent to end users and frame rates approaching ten frames per second can be achieved.

3.3 User Interface Concepts

VisIt has five primary user interface concepts:

Type	Description	Number
Database	How to read from a file	~115
Operator	How to manipulate data	~50
Plot	How to render data	~15
Expression	How to create new derived quantities	~190
Queries	How to extract quantitative info and debugging info	~90

A strength of VisIt's user interface concepts is its interoperability. Each plot can work on data directly from a file (databases) or from derived data (expressions), and can have an arbitrary number of data transformations or subselections applied (operators). Once the key information is extracted, quantitative or debugging information can be extracted (queries) or the data can be rendered (plots). Consider an example: a user reads from a file (database), calculates the λ-2 metric for finding high vorticity (expressions), isolates out the regions of highest vorticity (operators), renders it (plots), then calculates the number of connected components and statistics about them (queries).

VisIt makes it easy to add new types of databases, operators, and plots. The base infrastructure deals with these concepts as abstract types; it only discovers the concrete databases, operators, and plots at startup, by loading them as plugins. Developing new functionality translates to developing a new plugin. Further, VisIt aids the development process. It provides an environment for defining a plugin and then performs code generation. After the developer sets up the options for the plugins, VisIt generates attributes for storing the options, user interface components (Python, Qt, and Java), the plugin bindings, and C++ methods with "dummy" implementations. The developer then replaces the dummy implementations with their intended algorithm, file reading code, etc.

3.4 The size and breadth of VisIt

Although it is not discussed in depth in this paper, VisIt has an extensive list of features. Its ˜115 file format readers include support for many HDF5- and NetCDF-based formats, CGNS, and others, including generic readers for some types of binary and ASCII files. Its ˜60 operators include transformations such as projections, scaling, rotation, and translation, coordinate transformations, data subsetting, such as thresholding and contouring, and spatial thresholding, such as limiting to a box or a plane, among many others. Its ˜90 queries allow users to get customizable reports about specific cells or points, integrate quantities, calculate surface areas and volumes, insert synthetic diagnostics/virtual detectors, and much more. Its ˜190 expressions go well beyond simple math. For example, one can create derived quantities like "if the magnitude of the gradient of density is greater than this, then do this, else do that."

And many features do not fit into the five primary user interface concepts. There is support for positioning light sources, making movies (including MPEG encoding), eliminating data based on known categorizations (e.g. "show me only this refinement level" from an AMR mesh), and having plots cast a shadow, just to name a few. In total, VisIt is approximately one and a half million lines of code. Further, VisIt is built on top of many third party libraries, including the Visualization ToolKit (VTK), which contains many additional visualization algorithms.

4 Successes

The VisIt project has produced several forms of success: providing a scalable infrastructure for visualization and analysis, populating that infrastructure with cutting-edge algorithms, informing the limits of new hardware architectures, and, most importantly, enabling successes for the tool's end users. We summarize some of the most noteworthy highlights in the subsections below.

4.1 Scalability successes

In 2009, a pair of studies were run to demonstrate VisIt's capabilities for scalability and large data. In the first study, VisIt's infrastructure and some of its key visualization algorithms were demonstrated to be weakly scalable (see Figure 2). This demonstration led to be VisIt being selected as a "Joule code," a formal certification process by the US Office of Management and Budget to ensure that programs running on high end supercomputers are capable of using the machine efficiently. In the second study, VisIt was scaled up to tens of thousands of cores and used to visualize data sets with trillions of cells per time slice (see Figure 3). This study found VisIt itself to perform quite well, although overall performance was limited by the supercomputer's I/O bandwidth. Both studies are further described in [5].

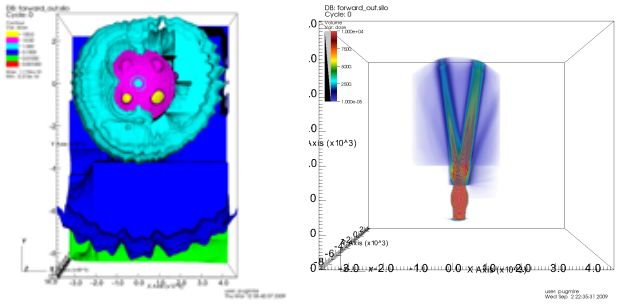

Figure 2: Contouring and volume rendering from a Denovo radiation transport simulation, produced by VisIt using 12,270 cores of JaguarPF as part of the "Joule code" certification, which showed that VisIt is weakly scalable.

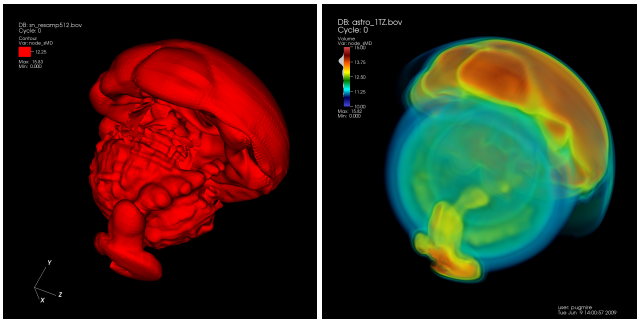

Figure 3: Contouring and volume rendering of a two trillion cell data set produced by VisIt using 32,000 cores of JaguarPF as part of a study on scalability at high levels of concurrency and on large data sets. The volume rendering was reproduced in 2011 on a one trillion cell version of the data set using only 800 cores of the TACC Longhorn machine.

4.2 A repository for large data algorithms

Many advanced algorithms for visualizing and analyzing large data have been implemented inside of VisIt, making them directly available to end users. Notable algorithms include:

- A novel streamline algorithm that melds two different parallelization strategies ("over data" and "over seeds") to retain their positive effects while minimizing their negative ones [13];
- A volume rendering algorithm that handles the compositing complexities inherent to unstructured meshes while still delivering scalable performance [3];
- An algorithm for identifying connected components in unstructured meshes in a distributed memory parallel setting on very large data sets [8];
- An algorithm for creating crack-free isosurfaces for adaptive mesh refinement data, a common mesh type for very large data [16].
- A well-performing material interface reconstruction algorithm for distributed memory parallel environments that balances concerns for both visualization and analysis [11]; and

- A method for repeated interpolations of velocity fields in unstructured meshes, to accelerate streamlines [7].

Further, VisIt has been the subject of much systems research, including papers on the base VisIt architecture [4], VisIt's "contract" system which allows it to detect the processing requirements for the current operations and adaptively apply the best optimizations [2], and a description of the adapter layer that allows VisIt to couple with a simulation and run *in situ* [17].

4.3 Supercomputing research performed with VisIt

As the landscape for parallel computers changes, VisIt has been used to test the benefits of emerging algorithms and hardware features, including:

- Studying modifications to collective communication patterns for ghost data generation to be suitable for out-of-core processing, thereby improving cache coherency and reducing memory footprint [9];
- Studying the viability of hardware accelerated volume rendering on distributed memory parallel visualization clusters powered by GPUs [6];
- Studying the benefits of hybrid parallelism for streamline algorithms [1]; and
- Studying the issues and strategies for porting to new operating systems [12].

4.4 User successes

Of course, the most important measure for the project is helping users better understand their data. Unfortunately, metrics in this space are difficult:

- Some national laboratories keep statistics on their user community: the United States laboratory Lawrence Livermore has approximately 300 regular users, the United Kingdom's Atomic Weapons Establishment (AWE) has approximately 100 regular users, and France's Atomic Energy Commission (CEA) at CESTA has approximately 50 regular users. Other laboratories, like Oak Ridge and Lawrence Berkeley, view VisIt as their primary visualization and analysis tool, but don't keep user statistics.
- In terms of monetary support for developing VisIt, the US Department of Energy funds VisIt development through its Office of Science, National Nuclear Security Agency, and Office of Nuclear Energy. Both of the US National Science Foundation (NSF) XD centers on visualization actively deploy and support VisIt as well.
- Another method for measuring usage is studying affiliations of users who ask questions on the mailing list. The majority of these inquiries come from none of the previously mentioned institutions, indicating that usage goes beyond these sites.

While images from VisIt are regularly used without citation, there have been several notable instances of publications using VisIt to perform novel analysis:

- Analysis of laser wakefield simulations often amounts to finding key particles. In [14], query-driven visualization techniques are used to search through terabytes of data to locate these key particles in as little as two seconds.
- Simulations often deal with idealized meshes. In [10], VisIt's comparative capabilities are used to quantify the importance of engineering defects when differencing as-built and as-designed models.
- The toroidal magnetic fields found in tokamaks are analyzed by studying flow through a cross-section and the topological "islands" this flow traces out. In [15], the authors describe how they perform this analysis using VisIt's streamline code.

5 Future Challenges

Although VisIt is well suited for today's supercomputing environment, the project will face many challenges in the future. In the short term, I/O limitations will force visualization and analysis activities to de-emphasize I/O. The VisIt development team has invested in pertinent techniques, such as multi-resolution processing and in situ, but these techniques will need to be further hardened to support regular production use. In the longer term, power limits will constrain data movement, forcing much processing to occur in situ on novel architectures, such as GPU accelerators. Unfortunately, VisIt's existing in situ implementation may be mismatched for this many-core future, for two reasons. First, although VisIt can be easily multi-threaded using a pthreads or OpenMP-type approach, this approach may not be able to take advantage of these architectures. The many-core future may require CUDA or OpenCL-type languages; migrating the VisIt code base to this setting would be a substantial undertaking. Second, although VisIt has been demonstrated to work well at high levels of concurrency, some of its algorithms involve large data exchanges. Although these algorithms perform well on current machines, they would violate the data movement constraints on future machines and would need to be re-designed.

6 Summary

The VisIt project's three focal points – understanding data, large data, and delivering a product – together form a powerful environment for analyzing data from HPC simulations. It is used in varied ways: it enables visualization scientists, computational code developers, and the physicists that run these codes to perform a broad range of data understanding activities, including debugging, making movies, and exploring data. The user interface portion of its design provides a powerful paradigm for analyzing data while the data processing portion of its design is well suited for big data. This in turn has led to many successes: in scaling up to high levels of concurrency and large data sizes, in providing a "home" for large data algorithms, in understanding how to best use supercomputers, and, most importantly, in helping users understand their data. Further, despite significant upcoming changes in supercomputing architecture, VisIt's future appears bright, as it enjoys vibrant user and developer communities.

Acknowledgements

This work was supported by the Director, Office of Science, Office and Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET).

References

[1] D. Camp, C. Garth, H. Childs, D. Pugmire, and K. I. Joy. Streamline integration using MPI-hybrid parallelism on a large multi-core architecture. *IEEE Transactions on Visualization and Computer Graphics*, 2011 (to appear).

[2] H. Childs, E. S. Brugger, K. S. Bonnell, J. S. Meredith, M. Miller, B. J. Whitlock, and N. Max. A Contract-Based System for Large Data Visualization. In *Proceedings of IEEE Visualization*, pages 190–198, 2005.

[3] H. Childs, M. Duchaineau, and K.-L. Ma. A scalable, hybrid scheme for volume rendering massive data sets. In *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 153–162, May 2006.

[4] H. Childs and M. Miller. Beyond meat grinders: An analysis framework addressing the scale and complexity of large data sets. In *SpringSim High Performance Computing Symposium (HPC 2006)*, pages 181–186, 2006.

[5] H. Childs, D. Pugmire, S. Ahern, B. Whitlock, M. Howison, Prabhat, G. Weber, and E. W. Bethel. Extreme Scaling of Production Visualization Software on Diverse Architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010. LBNL-3403E.

[6] T. Fogal, H. Childs, S. Shankar, J. Krüger, R. D. Bergeron, and P. Hatcher. Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *Proceedings of High Performance Graphics 2010*, pages 57–66, June 2010.

[7] C. Garth and K. Joy. Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Computer Graphics and Visualization*, 16(6):1541–1550, Nov. 2010.

[8] C. Harrison, H. Childs, and K. P. Gaither. Data-parallel mesh connected components labeling and analysis. In *EGPGV*, pages 131–140, 2011.

[9] M. Isenburg, P. Lindstrom, and H. Childs. Parallel and streaming generation of ghost data for structured grids. *IEEE Computer Graphics and Applications*, 30(3):32–44, 2010.

[10] E. J. Kokko, H. E. Martz, D. J. Chinn, H. R. Childs, J. A. Jackson, D. H. Chambers, D. J. Schneberk, and G. A. Clark. As-built modeling of objects for performance assessment. *Journal of Computing and Information Science in Engineering*, 6(4):405–417, 12 2006.

[11] J. S. Meredith and H. Childs. Visualization and analysis-oriented reconstruction of material interfaces. *Comput. Graph. Forum*, 29(3):1241–1250, 2010.

[12] D. Pugmire, H. Childs, and S. Ahern. Parallel Analysis and Visualization on Cray Compute Node Linux. In *Cray Users Group Meeting*, 2008.

[13] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. Weber. Scalable Computation of Streamlines on Very Large Datasets. In *Proceedings of Supercomputing*, 2009.

[14] O. Ruebel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel. High performance multivariate visual data exploration for extemely large data. In *SuperComputing 2008 (SC08)*, 2008. (accepted for publication).

[15] A. Sanderson, G. Chen, X. Tricoche, D. Pugmire, S. Kruger, and J. Breslau. Analysis of recurrent patterns in toroidal magnetic fields. In *Proceedings Visualization / Information Visualization 2010*, volume 16 of *IEEE Transactions on Visualization and Computer Graphics*, 2010.

[16] G. H. Weber, V. E. Beckner, H. Childs, T. J. Ligocki, M. Miller, B. van Straalen, and E. W. Bethel. Visualization tools for adaptive mesh refinement data. In W. Benger, R. Heinzel, W. Kapferer, W. Schoor, M. Tyagi, S. Venkataraman, and G. H. Weber, editors, *Proceedings of the 4th High End Visualization Workshop*, pages 12–25, Berlin, Germany, 2007. Lehmans Media. LBNL-62954.

[17] B. Whitlock, J.-M. Favre, and J. S. Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *EGPGV*, pages 101–109, 2011.